Polyhedral Compilation

Sven Verdoolaege

Computer Science, KU Leuven and Polly Labs

March 29, 2018

Outline

Motivation

- Computer architectures are becoming more difficult to program efficiently
  - multiple levels of parallelism
  - non-uniform memory architectures

⇒ Advanced compiler optimizations are required

- hierarchical partitioning and reordering of operations (e.g., parallelization, loop fusion, . . .)
- mapping to different processing units
- memory transfers between processing units

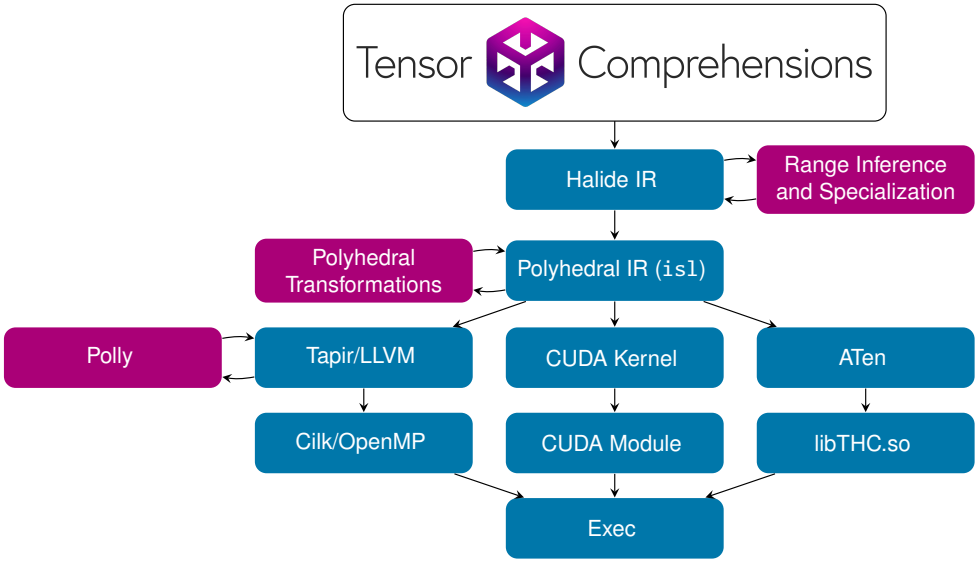
⇒ Global view of individual operations is required

⇒ Polyhedral Model

Alternatives are available, e.g., using rewrite rules as in Lift [17]

Polyhedral Compilation — Example Framework

[35]



Polyhedral Compilation — Example Framework

[35]

Tensor comprehensions inspired by Einstein notation

For example, matrix multiplication $C = A\,B$ with $A \in \mathbb{R}^{M \times K}$ and $B \in \mathbb{R}^{K \times N}$

$$C_{ij} = \sum A_{ik} B_{kj}$$

size of output inferred from input

defined rather than updated

```
C(i,j) +=! A(i,k) * B(k,j)
```

reduction over variables that only appear on RHS

Polyhedral model

- initialization instances
  `C(i,j) = 0` for $0 \le i < M \wedge 0 \le j < N$
- update instances
  `C(i,j) += A(i,k) * B(k,j)` for $0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K$

CUDA code generation from polyhedral model

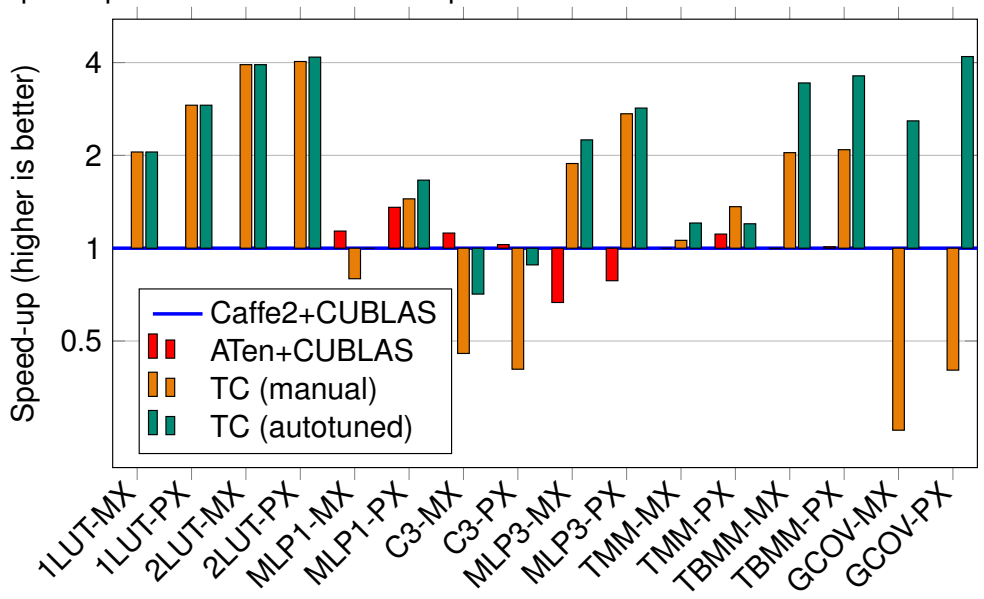
- tiling + full/partial tile separation
- mapping of instances to CUDA blocks/threads
- data copies to shared memory/registers
- mapping of full tiles to efficient library implementations

Polyhedral Compilation — Example Framework

[35]

TC Performance (Nov 2017):

Speedup of median run-time compared to Caffe2+CUBLAS



Polyhedral Model

[28]

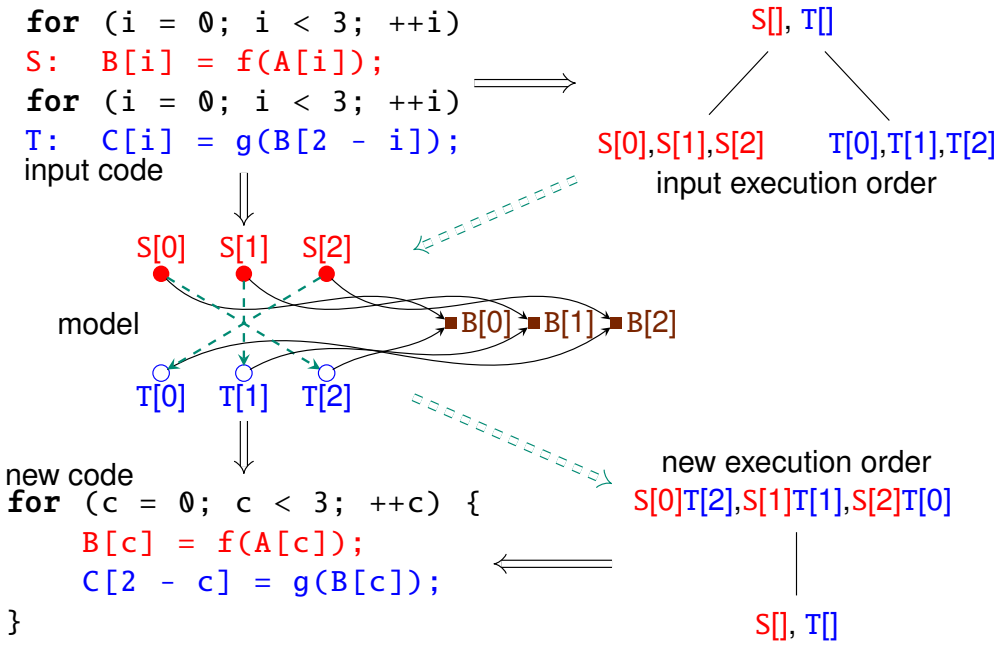
Key features

- instance based
  - ⇒ statement *instances*
  - ⇒ array *elements*
- compact representation based on polyhedra or similar objects
  - ⇒ Presburger sets and relations
  - ⇒ ...

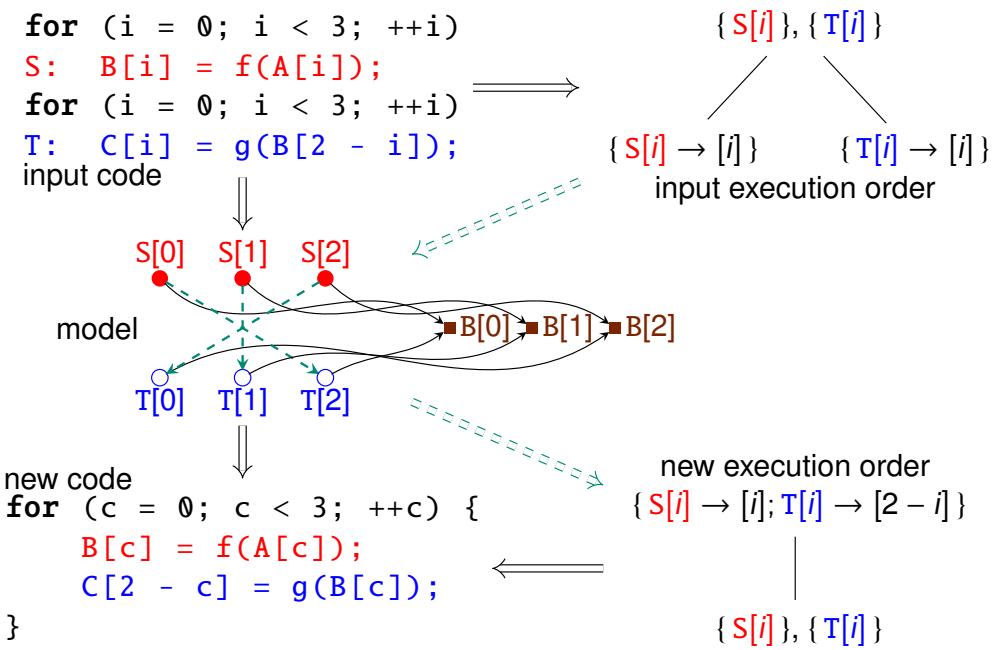
Main constituents of program representation

- Instance Set
  - ⇒ the set of all statement instances
- Access Relations
  - ⇒ the array elements accessed by a statement instance
- Dependences
  - ⇒ the statement instances that depend on a statement instance
- Schedule
  - ⇒ the relative execution order of statement instances
- Context
  - ⇒ constraints on parameters

Polyhedral Model — Example



Polyhedral Model — Example



Polyhedral Model

[28]

Key features

- instance based
  - ⇒ statement *instances*
  - ⇒ array *elements*
- compact representation based on polyhedra or similar objects
  - ⇒ Presburger sets and relations defined by Presburger formula
  - ⇒ ...
- quasi-affine expression (no multiplication)
  - variable x
  - constant integer number 3
  - constant symbol N
  - addition ($+$), subtraction ($-$) $x + 3$
  - integer division by integer constant d ($\lfloor \cdot / d \rfloor$) $\lfloor (x + 3)/16 \rfloor$
- Presburger formula
  - true
  - quasi-affine expression
  - less-than-or-equal relation ($\leq$) $0 \leq x$
  - equality ($=$)
  - first order logic connectives: $\wedge$, $\vee$, $\neg$, $\exists$, $\forall$ $0 \leq x \wedge x < N$

Quasi-affine Expressions and Presburger Formulas

- quasi-affine expression (no multiplication)
  - variable x
  - constant integer number 3
  - constant symbol N
  - addition ($+$), subtraction ($-$) $x + 3$
  - integer division by integer constant d ($\lfloor \cdot / d \rfloor$) $\lfloor (x + 3)/16 \rfloor$
- Presburger formula
  - true
  - quasi-affine expression
  - less-than-or-equal relation ($\leq$) $0 \leq x$
  - equality ($=$)
  - first order logic connectives: $\wedge$, $\vee$, $\neg$, $\exists$, $\forall$ $0 \leq x \wedge x < N$
- not allowed

  $x * x$, $x * N$
- allowed

  $3 * x \equiv x + x + x$

  $abs(x) \equiv [x \geq 0]x + [x < 0](-x)$

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Instance Set (set of statement instances)

$$\{ S1[i,j] : 0 \leq i < M \wedge 0 \leq j < N; \; S2[i,j,k] : 0 \leq i < M \wedge 0 \leq j < N \wedge 0 \leq k < K \}$$

- Access Relations (accessed array elements; W: write, R: read)

$$W = \{ S1[i,j] \to C[i,j]; S2[i,j,k] \to C[i,j] \}$$

$$R = \{ S2[i,j,k] \to C[i,j]; S2[i,j,k] \to A[i,k]; S2[i,j,k] \to B[k,j] \}$$

Schedule Representation [31]

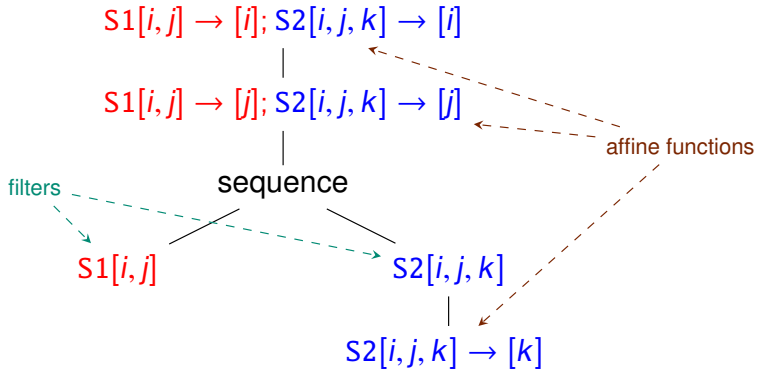
Schedule S keeps track of relative execution order of statement instances

⇒ for each pair of statement instances **i** and **j**, schedule determines
- **i** executed before **j** (**i** $<_S$ **j**),
- **i** executed after **j** (**j** $<_S$ **i**), or
- **i** and **j** may be executed simultaneously

Schedule trees form a combined hierarchical schedule representation
- Main constructs:
  - affine schedule: instances are executed according to affine function
  - *sequence*: partitions instances through child *filters* executed in order
- Order of instances determined by outermost node separating them
- Deriving schedule tree from AST
  - **for** loop ⇒ affine schedule corresponding to loop iterator
  - compound statement ⇒ sequence

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```



Schedule Representation [31]

Schedule S keeps track of relative execution order of statement instances

⇒ for each pair of statement instances **i** and **j**, schedule determines
- **i** executed before **j** (**i** $<_S$ **j**),
- **i** executed after **j** (**j** $<_S$ **i**), or
- **i** and **j** may be executed simultaneously

Schedule trees form a combined hierarchical schedule representation
- Main constructs:
  - affine schedule: instances are executed according to affine function
  - *band*: nested sequence of affine functions called its *members*; combined multi-dimensional affine function is called the *partial schedule* of the band
  - *sequence*: partitions instances through child *filters* executed in order
- Order of instances determined by outermost node separating them
- Deriving schedule tree from AST
  - **for** loop ⇒ affine schedule corresponding to loop iterator
  - compound statement ⇒ sequence

Named Presburger Relation Schedules

Schedule tree with single (band) node

Flattening a schedule tree
- two nested band nodes
  - ⇒ replace by single band node with concatenated partial schedule
- sequence with as children either leaves or trees consisting of a single band node
  - ⇒ treat leaves as zero-dimensional band nodes
  - ⇒ pad lower-dimensional bands (e.g., with zero)
  - ⇒ construct one-dimensional band assigning increasing values to children
  - ⇒ combine one-dimensional band with children

Parametric Example: Matrix Multiplication

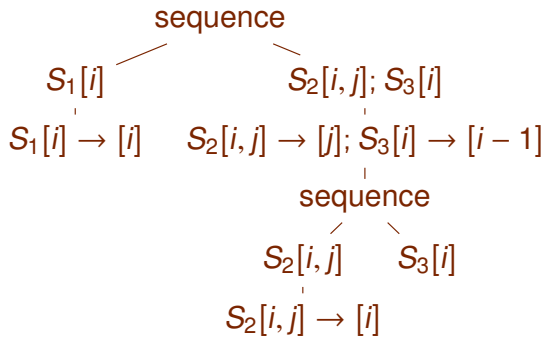
```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

$S1[i,j] \to [i,j,0,0]; S2[i,j,k] \to [i,j,1,k]$

Schedule Representation

[26, 31]

$$\{ S_1[i] \to [0,i,0,0]; \quad S_2[i,j] \to [1,j,0,i]; \quad S_3[i] \to [1,i-1,1,0] \}$$



- flat (union map) schedule
  - single object
  - schedule transformations can be composed as Presburger relations
  - flat schedule space (padding)
- schedule tree
  - single object
  - structured schedule space
  - tailored schedule tree transformations

Loop Transformations and the Polyhedral Model

Loop transformations result in
different execution order of statement instances
$\Rightarrow$ different schedule

Polyhedral model can be used to

- evaluate a schedule and/or
- construct a schedule

Polyhedral schedules can represent (combinations of)

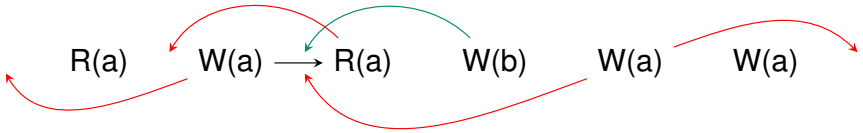
- loop distribution
- loop fusion
- loop tiling
- ...

Schedule Properties

- Validity
  New schedule should preserve meaning
- Parallelism
  Can the iterations of a given loop be executed in parallel?
- Locality
  Statement instances scheduled closely to each other
- Tilability
  Can a given schedule band be tiled?
- Consecutivity
  Are elements of array accessed consecutively?

Schedule Validity [2]

New schedule should preserve meaning



Internal restrictions

- No read of a value may be scheduled before the write of the value
- No other write to same memory location may be scheduled in between

External restrictions (on non-temporaries)

- No write may be scheduled before initial read from a memory location
- No write may be scheduled after last write to a memory location

Sufficient conditions:

- Every read of a memory location is scheduled after every preceding write to the same memory location
- Every write to a memory location is scheduled after every preceding read or write to the same memory location

Dependences

Sufficient conditions for validity of schedule S:

- Every read of a memory location is scheduled after every preceding write to the same memory location
- Every write to a memory location is scheduled after every preceding read or write to the same memory location

Dependence relation D: pairs of statement instances

- accessing the same memory location
- of which at least one is a write
- with the first executed before the second in original code

Sufficient condition:

$$\forall \mathbf{i} \to \mathbf{j} \in D : \mathbf{i} <_S \mathbf{j}$$

Local Validity

Schedule validity:

$$\forall \mathbf{i} \to \mathbf{j} \in D : \mathbf{i} <_S \mathbf{j}$$

Consider subset of *local* dependences L

At outermost node: $L = D$

Current node

- band node with partial schedule f

  $$\forall \mathbf{i} \to \mathbf{j} \in L : f(\mathbf{i}) \leq_{lex} f(\mathbf{j})$$

  Carried dependences: $\mathbf{i} \to \mathbf{j} \in L : f(\mathbf{i}) \neq f(\mathbf{j})$
  $\Rightarrow$ no longer need to be considered in nested nodes
  Remaining dependences: $L' = \{ \mathbf{i} \to \mathbf{j} \in L : f(\mathbf{i}) = f(\mathbf{j}) \}$
- sequence node with child position p and filters F_k

  $$\forall \mathbf{i} \to \mathbf{j} \in L : p(\mathbf{i}) \leq p(\mathbf{j})$$

  Carried dependences: $\mathbf{i} \to \mathbf{j} \in L : p(\mathbf{i}) \neq p(\mathbf{j})$
  Remaining dependences in child c: $L' = \{ \mathbf{i} \to \mathbf{j} \in L : \mathbf{i}, \mathbf{j} \in F_c \}$
- leaf node: $L = \emptyset$

Loop Distribution Validity

```
for (int i = 1; i < 100; ++i) {
S:      A[i] = f(i);
T:      B[i] = A[i] + A[i - 1];
}
```

$\{ S[i] \to [i]; T[i] \to [i] \}$

$\{ S[i] \}, \{ T[i] \}$

Dependences:

$$\{ S[i] \to T[i] : 1 \leq i < 100; S[i] \to T[i+1] : 1 \leq i, i+1 < 100 \}$$

$\{ S[i] \to [i]; T[i] \to [i] \}$
satisfied: $\{ S[i] \to T[i] : 1 \leq i < 100; S[i] \to T[i+1] : 1 \leq i, i+1 < 100 \}$
carried: $\{ S[i] \to T[i+1] : 1 \leq i, i+1 < 100 \}$
$\{ S[i] \}, \{ T[i] \}$
satisfied: $\{ S[i] \to T[i] : 1 \leq i < 100 \}$
carried: $\{ S[i] \to T[i] : 1 \leq i < 100 \}$

Loop Distribution Validity

```
for (int i = 1; i < 100; ++i) {
S:      A[i] = f(i);
T:      B[i] = A[i] + A[i - 1];
}
```

$\{ S[i] \to [i]; T[i] \to [i] \}$

$\{ S[i] \}, \{ T[i] \}$

Dependences:

$$\{ S[i] \to T[i] : 1 \le i < 100; S[i] \to T[i+1] : 1 \le i, i+1 < 100 \}$$

Loop distribution

```
for (int i = 1; i < 100; ++i)
        A[i] = f(i);
for (int i = 1; i < 100; ++i)
        B[i] = A[i] + A[i - 1];
```

$\{ S[i] \}, \{ T[i] \}$

$\{ S[i] \to [i] \}$ $\{ T[i] \to [i] \}$

$\{ S[i] \}, \{ T[i] \}$
satisfied: $\{ S[i] \to T[i] : 1 \le i < 100; S[i] \to T[i+1] : 1 \le i, i+1 < 100 \}$
carried: $\{ S[i] \to T[i] : 1 \le i < 100; S[i] \to T[i+1] : 1 \le i, i+1 < 100 \}$

Polyhedral Model Schedule Properties March 29, 2018 25 / 77

Loop Distribution Validity

```
for (int i = 1; i < 100; ++i) {
S:      A[i] = f(i);
T:      B[i] = A[i] + A[i + 1];
}
```

$\{ S[i] \to [i]; T[i] \to [i] \}$

$\{ S[i] \}, \{ T[i] \}$

Dependences:

$$\{ S[i] \to T[i] : 1 \le i < 100; T[i] \to S[i+1] : 1 \le i, i+1 < 100 \}$$

Loop distribution

```
for (int i = 1; i < 100; ++i)
        A[i] = f(i);
for (int i = 1; i < 100; ++i)
        B[i] = A[i] + A[i + 1];
```

$\{ S[i] \}, \{ T[i] \}$

$\{ S[i] \to [i] \}$ $\{ T[i] \to [i] \}$

$\{ S[i] \}, \{ T[i] \}$
satisfied: $\{ S[i] \to T[i] : 1 \le i < 100 \}$
violated: $\{ T[i] \to S[i+1] : 1 \le i, i+1 < 100 \}$

Parallel Loops and Parallel Band Members

Recall:
Iterations of a given loop can be executed in parallel if
writes of iteration do not conflict with reads/writes of other iteration
iff there is no dependence between distinct iterations
(for any given iteration of the outer loops)

A band member with affine function f is parallel if

$$\forall \mathbf{i} \to \mathbf{j} \in L : f(\mathbf{i}) = f(\mathbf{j})$$

with L the local dependences

Polyhedral Model Schedule Properties March 29, 2018 28 / 77

Loop Distribution and Parallelism

```
for (int i = 1; i < 100; ++i) {
S:      A[i] = f(i);
T:      B[i] = A[i] + A[i - 1];
}
```

$\{ S[i] \to [i]; T[i] \to [i] \}$

$\{ S[i] \}, \{ T[i] \}$

Dependences:

$$\{ S[i] \to T[i] : 1 \le i < 100; S[i] \to T[i+1] : 1 \le i, i+1 < 100 \}$$

$\{ S[i] \to [i]; T[i] \to [i] \}$
local: $\{ S[i] \to T[i] : 1 \le i < 100; S[i] \to T[i+1] : 1 \le i, i+1 < 100 \}$
conflict: $\{ S[i] \to T[i+1] : 1 \le i, i+1 < 100 \}$
$\Rightarrow$ not parallel

Loop Distribution and Parallelism

```
for (int i = 1; i < 100; ++i) {
S:      A[i] = f(i);
T:      B[i] = A[i] + A[i - 1];
}
```

$\{ S[i] \to [i]; T[i] \to [i] \}$

$\{ S[i] \}, \{ T[i] \}$

Dependences:

$$\{ S[i] \to T[i] : 1 \le i < 100; S[i] \to T[i+1] : 1 \le i, i+1 < 100 \}$$

Loop distribution

```
for (int i = 1; i < 100; ++i)
        A[i] = f(i);
for (int i = 1; i < 100; ++i)
        B[i] = A[i] + A[i - 1];
```

$\{ S[i] \}, \{ T[i] \}$

$\{ S[i] \to [i] \}$ $\{ T[i] \to [i] \}$

$\{ S[i] \to [i] \}$
local: $\emptyset$
conflict: $\emptyset$
$\Rightarrow$ parallel

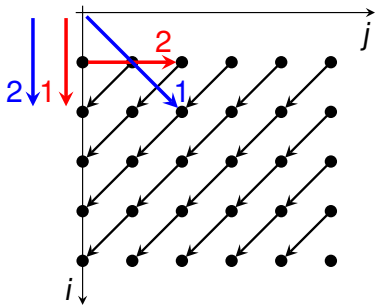
$\{ T[i] \to [i] \}$
local: $\emptyset$
conflict: $\emptyset$
$\Rightarrow$ parallel

Parallelism Example

```
for (int i = 1; i < 6; ++i)
    for (int j = 0; j < 6; ++j)
S:      A[i][j] = f(A[i - 1][j + 1]);
```

Dependences:

$$\{ S[i,j] \to S[i+1, j-1] : 1 \le i, i+1 < 6 \wedge 0 \le j, j-1 < 6 \}$$



original schedule:
$S[i,j] \to [i,j]$
new schedule:
$S[i,j] \to [i+j, i]$
$(i+j)$-direction is outer parallel

Decomposition: loop skewing + loop interchange

$[i,j] \to [i, i+j] \to [i+j, i]$

Locality

Statement instances **i** and **j** that reuse memory
$\Rightarrow$ scheduled closely to each other: $f(\mathbf{j}) - f(\mathbf{i})$ small

Types of locality:

- temporal locality
  $\Rightarrow$ instances that access the same memory element
- spatial locality
  $\Rightarrow$ instances that access adjacent memory elements

Sometimes further distinction made:

- self locality
  $\Rightarrow$ pair of instances from same statement
- group locality
  $\Rightarrow$ any pair of statement instances

Temporal locality often restricted to
pairs of writes and reads that refer to the same value
$\Rightarrow$ dataflow

Tiling a Band

Input:

- band of affine schedule functions
  $$f_1, f_2, \ldots, f_n$$
- tile sizes
  $$T_1, T_2, \ldots, T_n$$

Steps (conceptually)

1. divide each direction into chunks of size T_i (strip-mining)
   $$\lfloor f_1/T_1 \rfloor, f_1, \lfloor f_2/T_2 \rfloor, f_2, \ldots, \lfloor f_n/T_n \rfloor, f_n$$
   does not change execution order $\Rightarrow$ always valid
2. combine the chunking (interchange)
   $$\lfloor f_1/T_1 \rfloor, \lfloor f_2/T_2 \rfloor, \ldots, \lfloor f_n/T_n \rfloor, f_1, f_2, \ldots, f_n$$
   sufficient condition for interchange:
   all members are valid for local dependences at (top of) band
   $\Rightarrow$ permutable band

Loop Tiling Example

```
for (int i = 0; i < 8; ++i)
    for (int j = 0; j < 8; ++j)
S:      C[i][j] = A[i] * B[j];
```

1. strip-mine

$$S[i,j] \to 4\lfloor i/4 \rfloor$$
$$S[i,j] \to i$$
$$S[i,j] \to 4\lfloor j/4 \rfloor$$
$$S[i,j] \to j$$

```
for (int ti = 0; ti < 8; ti += 4)
    for (int i = ti; i < ti + 4; ++i)
        for (int tj = 0; tj < 8; tj += 4)
            for (int j = tj; j < tj + 4; ++j)
                C[i][j] = A[i] * B[j];
```


Loop Tiling Example

```
for (int i = 0; i < 8; ++i)
    for (int j = 0; j < 8; ++j)
S:      C[i][j] = A[i] * B[j];
```

1. strip-mine
2. interchange

$$S[i,j] \to 4\lfloor i/4 \rfloor$$
$$S[i,j] \to 4\lfloor j/4 \rfloor$$
$$S[i,j] \to i$$
$$S[i,j] \to j$$

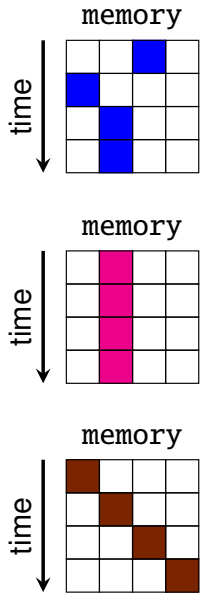
```
for (int ti = 0; ti < 8; ti += 4)
    for (int tj = 0; tj < 8; tj += 4)
        for (int i = ti; i < ti + 4; ++i)
            for (int j = tj; j < tj + 4; ++j)
                C[i][j] = A[i] * B[j];
```


Consecutivity Concept

Spatial Locality
Consecutive operations access neighboring memory elements

⇒ reuse of cache lines

- Temporal Locality
  Consecutive operations access the same memory element
  ⇒ reuse of data in cache or registers
- Consecutivity
  Consecutive operations access consecutive memory elements
  ⇒ vectorization
  ⇒ hardware cache prefetcher
  ⇒ burst accesses, e.g., on FPGA (Xilinx)

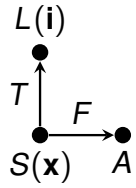


Consecutivity Criterion

Consecutive operations access consecutive memory elements

Assume (for the purpose of consecutivity)

- intra-statement consecutivity (⇒ per statement)
- row-major array layout
- purely affine access function
- purely affine per-statement schedule



Transformed access function FT^{-1} exhibits consecutivity if

- outer index expressions independent of innermost loop iterator
- innermost index expression proportional to innermost loop iterator

$$[\ldots + 0i_n]\ldots[\ldots + 0i_n][\ldots + 1i_n]$$

$$FT^{-1} = \begin{bmatrix} M & \mathbf{0} \\ N & 1 \end{bmatrix}$$

Consecutivity Criterion and Spatial Locality

Spatial Locality

$$F\,T^{-1} = \begin{bmatrix} M & \mathbf{0} \\ N & x \end{bmatrix}$$

- Temporal Locality

$$F\,T^{-1} = \begin{bmatrix} M & \mathbf{0} \\ N & 0 \end{bmatrix}$$

- Consecutivity

$$F\,T^{-1} = \begin{bmatrix} M & \mathbf{0} \\ N & 1 \end{bmatrix}$$

Operations on Polyhedral Model

- Model Extraction
  - Input: AST
  - Output: instance set, access relations, original schedule
- Dependence analysis
  - Input: instance set, access relations, original schedule
  - Output: dependence relations
- Scheduling
  - Input: instance set, dependence relations
  - Output: schedule
- AST generation (polyhedral scanning, code generation)
  - Input: instance set, schedule
  - Output: AST
- Data layout transformations
  - Input: access relations, dependence relations
  - Output: transformed access relations

Polyhedral Model Requirements

Requirements for basic polyhedral model: “regular” code

- Static control
  ⇒ control does not depend on input data
- Affine
  ⇒ all relevant expressions are (quasi-)affine
- No Aliasing
  ⇒ essentially no pointer manipulations

Note:

- polyhedral model may be *approximation* of input that does not strictly satisfy all requirements
- many *extensions* are available

Aliasing

[1]

Some possible ways of handling aliasing:

- use an input language that does not permit aliasing
- assume there is no aliasing
- require user to ensure absence of aliasing
  ⇒ e.g., use `restrict` keyword (in C)
- handle as may-write
  ⇒ may lead to too many dependences
- check aliasing at run-time
  ⇒ use original code in case of aliasing

Dependence Analysis

Recall: sufficient conditions for validity of schedule S:

$$\forall \mathbf{i} \rightarrow \mathbf{j} \in D : \mathbf{i} <_S \mathbf{j}$$

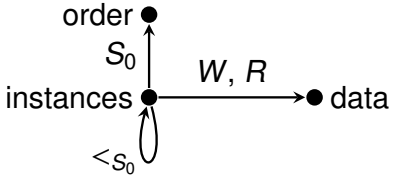
Dependence relation D: pairs of statement instances

- accessing the same memory location
- of which at least one is a write
- with the first executed before the second in original code

Computation:

$$D = \left(\left(W^{-1} \circ R\right) \cup \left(W^{-1} \circ W\right) \cup \left(R^{-1} \circ W\right)\right) \cap (<_{S_0})$$

W: write access relation
R: read access relation
S_0: original schedule



False Dependences

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```

Dependences

- read-after-write ("true"): $\{ \mathtt{S}[i] \rightarrow \mathtt{T}[i'] : i' \geq i \}$
- dataflow (subset of "true" dependences): $\{ \mathtt{S}[i] \rightarrow \mathtt{T}[i] \}$
- write-after-read ("anti"): $\{ \mathtt{T}[i] \rightarrow \mathtt{S}[i'] : i' > i \}$
- write-after-write ("output"): $\{ \mathtt{S}[i] \rightarrow \mathtt{S}[i'] : i' > i \}$

} "false"

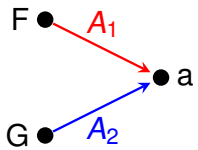
False dependences not from dataflow, but from reuse of memory location `t`

Array Dataflow Analysis

[14]

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
G:  g(a[i]);
```



Access relations:
$A_1 = \{ \mathtt{F}[i,j] \rightarrow \mathtt{a}[i+j] : 0 \leq i < N \wedge 0 \leq j < N - i \}$
$A_2 = \{ \mathtt{G}[i] \rightarrow \mathtt{a}[i] : 0 \leq i < N \}$
Map to all writes: $R'' = {A_1}^{-1} \circ A_2 = \{ \mathtt{G}[i] \rightarrow \mathtt{F}[i', i - i'] : 0 \leq i' \leq i < N \}$
Map to all preceding writes:
$R' = R'' \cap (<_S)^{-1} = \{ \mathtt{G}[i] \rightarrow \mathtt{F}[i', i - i'] : 0 \leq i' \leq i < N \}$
Last preceding write: $R = \max_{<_S} R' = \{ \mathtt{G}[i] \rightarrow \mathtt{F}[i, 0] : 0 \leq i < N \}$

Approximate Dataflow Analysis

How to compute dataflow in presence of data dependent control?

Two approaches

- Direct computation
  - distinguish between may- and must-writes
- Derived from exact run-time dependent dataflow
  - compute exact dataflow in terms of run-time information
  - exploit properties of run-time information
  - project out run-time information

May Writes

Keep track of whether write is possible or definite

- Must-writes
  Array elements are definitely written by statement instance
- May-writes
  Array elements are possibly written by statement instance
  - statement instance not necessarily executed
    ```
    for (i = 0; i < n; ++i)
        if (A[i] > 0)
    S:      B[i] = A[i];
    ```
    May-write: $\{ S[i] \to B[i] \}$
  - array element not necessarily accessed
    ```
    int A[N];
    /* ... */
    T:  A[B[0]] = 5;
    ```
    May-write: $\{ T[] \to A[a] : 0 \leq a < N \}$

Must-write access relation is subset of may-write access relation

Approximate Dataflow — Direct Computation

- Read-after-write dependences
  - write and read access same memory location
  - write executed before the read

  ⇒ Approximate dataflow analysis with no must-writes
- Dataflow dependences
  - write and read access same memory location
  - write executed before the read
  - no intermediate write to same memory location
    ⇒ intermediate write kills dependence
- Approximate dataflow dependences
  - may-write and read access same memory location
  - may-write executed before the read
  - no intermediate must-write to same memory location
    ⇒ intermediate must-write kills dependence

Dependence analysis in `isl` [28, 29]

`isl` contains generic dependence analysis engine
⇒ determines dependence relations between “sources” and “sinks”

Input:

- Sink $K : I \to D$
- May-source $Y : I \to D$
- Kill $L : I \to D$
- Schedule S on I ⇒ defines “before” and “intermediate”

Output:

- May-dependence relation: triples $(\mathbf{i}, \mathbf{k}, \mathbf{a})$
  - **i** has a may-source to **a**
  - **k** has a sink to **a**
  - **i** is scheduled before **k**
  - there is no intermediate kill to **a**
- May-no-source: sinks $\mathbf{k} \to \mathbf{a}$ with no kill to **a** before **k**

Dependence analysis in `PPCG` [29]

`isl`:

- May-dependence relation: triples $(\mathbf{i}, \mathbf{k}, \mathbf{a})$
  - **i** has a may-source to **a**
  - **k** has a sink to **a**
  - **i** is scheduled before **k**
  - there is no intermediate kill to **a**
- May-no-source: sinks $\mathbf{k} \to \mathbf{a}$ with no kill to **a** before **k**

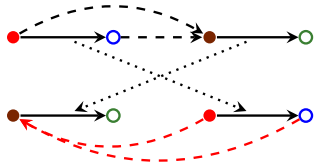
`PPCG` (without live-range reordering):

- flow dependences (without **a**) and live-in (may-no-source)
  - sink: may-read
  - may-source: may-write
  - kill: must-write
- false dependences (without **a**)
  - sink: may-write
  - may-source: may-read or may-write
  - kill: must-write
- killed writes (without **k**) (⇒ removed from may-write to get live-out)
  - sink: must-write
  - may-source: may-write

Live-Range Reordering

[27, 29]

```
a = f1();
f2(a);
a = f3();
f4(a);
```



⟶: flow
- - →: false

Reordering rejected due to false dependences

Live-range reordering

- allows such live-ranges to be reordered
- using somewhat different classification of dependences
- computed using different calls to the same dependence analysis engine

Approximate Dataflow Analysis

How to compute dataflow in presence of data dependent control?

Two approaches

- Direct computation
  - distinguish between may- and must-writes
- Derived from exact run-time dependent dataflow
  - compute exact dataflow in terms of run-time information
  - exploit properties of run-time information
  - project out run-time information

Run-time Dependent Dataflow Analysis

[5, 33]

Approaches

- "fuzzy array dataflow analysis"
- "on-demand-parametric array dataflow analysis"

```
for (int i = 0; i < n; ++i) {
S1:     t = f1(i);
S2:     A[i] = t;
S3:     t = f2(i);
S4:     if (f3(i))
S5:             t = f4(i);
S6:     B[i] = t;
}
```

- Run-time dependent dataflow
  $\{ S1[i] \rightarrow S2[i]; S3[i] \rightarrow S6[i] : \beta^{S5}_{S6} = 0; S5[i] \rightarrow S6[i] : \beta^{S5}_{S6} = 1 \}$
  β^P_C: any potential source instance P is executed for sink C
  λ^P_C: last potential source instance P executed for sink C
- Approximate dataflow (project out β and λ)
  $\{ S1[i] \rightarrow S2[i]; S3[i] \rightarrow S6[i]; S5[i] \rightarrow S6[i] \}$

Polyhedral Process Networks

[25]

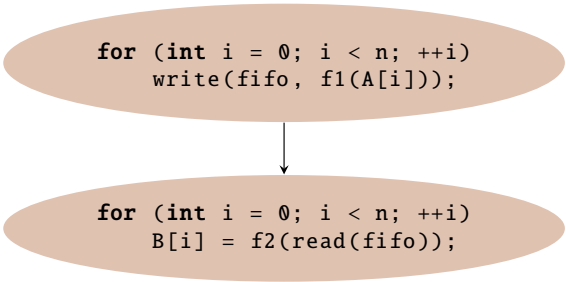
- Main purpose: extract task level parallelism from dataflow graph

statement	→	process
flow dependence	→	communication channel

  ⇒ requires dataflow analysis
- Processes are mapped to parallel hardware (e.g., FPGA)

Example:

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```

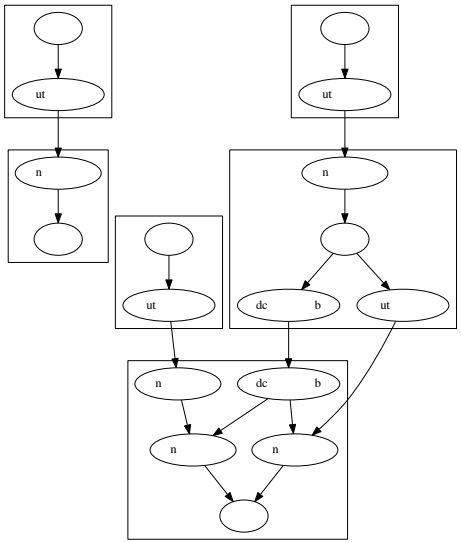


Process Networks with Dynamic Control

```
for (int i = 0; i < n; ++i) {
S1:     t = f1(i);
S2:     A[i] = t;
S3:     t = f2(i);
S4:     if (f3(i))
S5:             t = f4(i);
S6:     B[i] = t;
}
```

Run-time dependent dataflow:

$\{ S1[i] \rightarrow S2[i]; S3[i] \rightarrow S6[i] : \beta^{S5}_{S6} = 0;$
$S5[i] \rightarrow S6[i] : \beta^{S5}_{S6} = 1; S4[i] \rightarrow S5[i] \}$



Polyhedral Scheduling

[9, 10, 15, 22, 34]

Polyhedral model can be used to

- evaluate a schedule and/or
- construct a schedule

Types of Schedulers

- transitive closure based
  - consider slices of instances connected through dependences
  - execute slices in parallel
- Farkas based
  - one-shot scheduler
  - row-by-row scheduler
    - ★ Feautrier
    - ★ Pluto
- ...

Constraints on Schedule Coefficients

Affine schedule row:

$$f_i(\mathbf{x}) = \boxed{\mathbf{c}^{\mathrm{x}}_i}\boxed{\mathbf{x}} + \boxed{\mathbf{c}^{\mathrm{n}}_i}\boxed{\mathbf{n}} + \boxed{c^{\mathrm{c}}_i}$$

Validity constraints $S_i[\mathbf{x}] \rightarrow S_j[\mathbf{y}]$: $\quad f_j(\mathbf{y}) - f_i(\mathbf{x}) \geq 0$

Farkas → constraints on $\mathbf{c}^{\mathrm{x}}_i$, $\mathbf{c}^{\mathrm{n}}_i$ and c^{c}_i

- one-shot scheduler
  - ⇒ compute entire (flat) schedule using a single ILP
- row-by-row scheduler
  - ⇒ solve separate ILP for each row
  - ⇒ update constraints (and tree) after each row
  - Feautrier
    - ★ maximal inner parallelism
      - ⇒ carry as many dependences as possible at outer bands
  - Pluto
    - ★ tilable bands
    - ★ locality: $f(\mathbf{y}) - f(\mathbf{x})$ small
      - ⇒ parallelism as extreme case: $f(\mathbf{y}) - f(\mathbf{x}) = 0$

Data Layout Transformations

[12, 13]

- Memory compaction
  Reuse memory locations to store different data
  - ⇒ apply non-injective mapping to array elements
  - ⇒ reduce memory requirements
  - ⇒ extreme case: replace array by scalar

```
for (int i = 0; i < 100; ++i) {
        A[i] = f(i);
        B[i] = g(A[i]);
}
```

- Expansion
  Use different memory locations to store different data
  - ⇒ map different accesses to memory element to distinct locations
  - ⇒ increase scheduling freedom (e.g., more parallelism)

False Dependences

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```

Dependences

- read-after-write (“true”): $\{ S[i] \rightarrow T[i'] : i' \geq i \}$
- dataflow (subset of “true” dependences): $\{ S[i] \rightarrow T[i] \}$
- write-after-read (“anti”): $\{ T[i] \rightarrow S[i'] : i' > i \}$
- write-after-write (“output”): $\{ S[i] \rightarrow S[i'] : i' > i \}$

} “false”

False dependences not from dataflow, but from reuse of memory location `t`

Possible solution: expansion/privatization

```
for (int i = 0; i < n; ++i) {
S:      t[i] = f1(A[i]);
T:      B[i] = f2(t[i]);
}
```


Expansion

Assume:

- instance sets and access relations are static and exact
  ⇒ each read has exactly one corresponding write
- single read and write per statement
  ⇒ expanded array indexed by statement instance of write

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```

Dataflow: $\{ S[i] \rightarrow T[i] \}$

```
for (int i = 0; i < n; ++i) {
S:      S[i] = f1(A[i]);
T:      B[i] = f2(S[i]);
}
```

⇒ only remaining dependences are dataflow induced

Maximal Static Expansion [4]

```
for (int i = 0; i < n; ++i) {
S1:     t = f1(i);
S2:     A[i] = t;
S3:     t = f2(i);
S4:     if (f3(i))
S5:         t = f4(i);
S6:     B[i] = t;
}
```

```
t1[i] = f1(i);
A[i] = t1[i];
t2[i] = f2(i);
if (f3(i))
    t2[i] = f4(i);
B[i] = t2[i];
```

Dataflow cannot be determined independently of run-time information

⇒ approximate dataflow
$\{ S1[i] \rightarrow S2[i]; S3[i] \rightarrow S6[i]; S5[i] \rightarrow S6[i] \}$

⇒ a read may be associated to more than one write

⇒ corresponding equivalence classes should not be expanded apart

Polyhedral Software [3, 6, 7, 8, 10, 11, 16, 18, 19, 20, 21, 23, 24, 30, 32, 36]

http://polyhedral.info/software.html

- Core set manipulation libraries
  - integer sets: `isl`, `omega`, ...
  - rational sets: `PolyLib`, `PPL`, ...
- Model extraction
  - `clan`, `pet`, ...
- Dependence analysis
  - `petit`, `candl`, `isl`, `FADA`, ...
- Scheduler libraries
  - `LetSee`, `isl`, ...
- AST generation
  - `omega`, `CLooG`, `isl`, ...
- Source-to-source polyhedral compilers
  - `Pluto`, `PoCC`, `PPCG`, ...
- Compilers using polyhedral compilation
  - gcc/`graphite`, LLVM/`Polly`, ...

Cardinality

- Cardinality of a set
  - ⇒ number of elements in the set
  - ⇒ may depend on constant symbols

  $\operatorname{card} S = \{ n : n = \#S \}$

  $\operatorname{card} \{ \mathtt{A}[i] : 0 \le i \le n; \mathtt{B}[] \} = n + 2$
- Cardinality of a binary relation
  - ⇒ for each domain element, number of corresponding images

  $\operatorname{card} R = \{ \mathbf{i} \to n : n = \#(R(\{\mathbf{i}\})) \}$

  $R = \{ \mathtt{A}[i] \to \mathtt{C}[i] : 0 \le i \le n; \mathtt{B}[] \to \mathtt{C}[i] : 0 \le i \le n \}$

  $\operatorname{card} R = \{ \mathtt{A}[i] \to 1 : 0 \le i \le n; \mathtt{B}[] \to n + 1 \}$

⇒ not a Presburger formula

Cardinality Examples

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

  $\operatorname{card}\{ [i,j] : 0 \le i < N \wedge 0 \le j < N - i \}$

  ⇒ $\{ \frac{N+N^2}{2} : N \ge 1 \}$
- How many times is a given array element written?

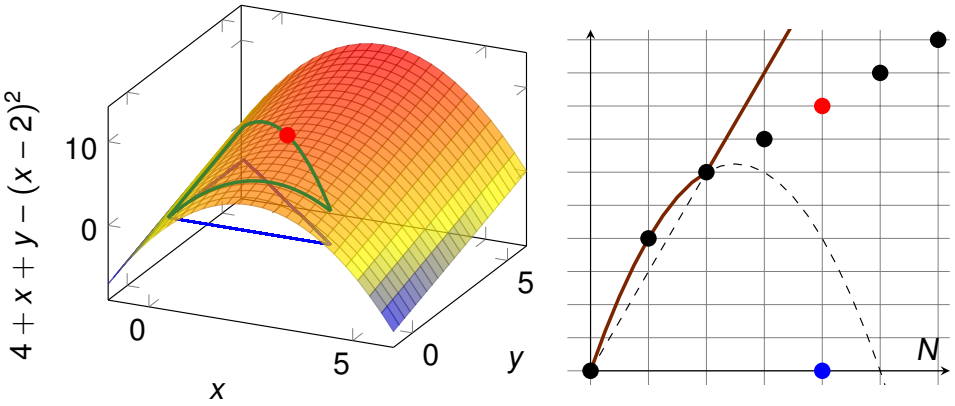
  $\operatorname{card}\left(\{ [i,j] \to \mathtt{a}[i+j] : 0 \le i < N \wedge 0 \le j < N - i \}^{-1} \right)$

  ⇒ $\{ \mathtt{a}[a] \to 1 + a : 0 \le a < N \}$
- How many array elements are written?

  $\operatorname{card}\left(\operatorname{ran} \{ [i,j] \to \mathtt{a}[i+j] : 0 \le i < N \wedge 0 \le j < N - i \} \right)$

  ⇒ $\{ N : N \ge 1 \}$

Bounds on Piecewise Quasi Polynomials



$$m(N) = \max_{(x,y):x,y\ge 0 \wedge x+y\le N} 4 + x + y - (x-2)^2 \le u(N) = \max(3N, 5N - N^2)$$

It may not be possible to compute exact maximum in all cases.

Upper bound $u(N) \ge m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

$$\operatorname{ub}\begin{cases} ij + i - N + 1 & \text{if } 0 \le i < N \wedge i \le j < N \end{cases}$$

Result:

$$\begin{cases} \max(1 - 2N + N^2) & \text{if } N \ge 1 \end{cases}$$

(exact maximum)

Incremental Counting

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

How many times is the statement executed?

- direct computation

  ```
  card [N] -> { [i,j] : 0 <= i < N and 0 <= j < N-i };
  ```

- incremental computation

  ```
  card [N] -> { [i] -> [j] : 0 <= i < N and 0 <= j < N-i };
  ```

  Result:

  ```
  [N] -> { [i] -> (N - i) : i <= -1 + N and i >= 0 }
  ```

  ```
  sum [N] -> { [i] -> (N - i) : i <= -1 + N and i >= 0 };
  ```

  ⇒ sum over all elements in domain

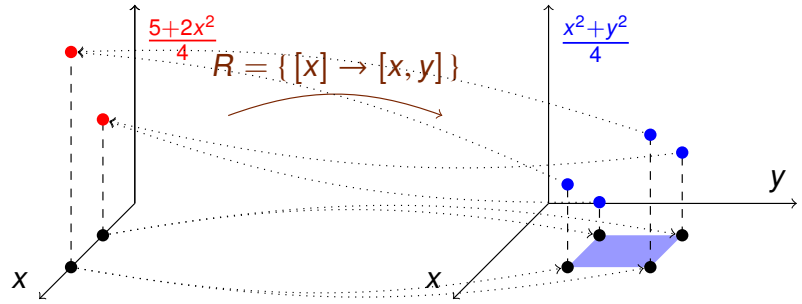
Weighted Counting

$$G = F \circ R = \left\{ [x,y] \to \frac{x^2+y^2}{4} : 1 \le x,y \le 2 \right\} \circ \{ [x] \to [x,y] \}$$

$$= \left\{ [x] \to \frac{5+2x^2}{2} : 1 \le x \le 2 \right\}$$

with F a piecewise quasi polynomial and R a Presburger relation is a piecewise quasi polynomial G such that

$$G(\mathbf{i}) = \sum_{\mathbf{j}:R(\mathbf{i},\mathbf{j})} F(\mathbf{j})$$



Example: Total Memory Allocation

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        p[i][j] = malloc(i * j + i - N + 1);
/* ... */
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        free(p[i][j]);
```

How much memory allocated in total?

$$F = \{ [i,j] \to ij + i - N + 1 \}$$

$$I = \{ [i,j] : 0 \le i < N \wedge i \le j < N \}$$

$$F(I) = \left\{ \frac{5}{12}N - \frac{1}{8}N^2 - \frac{5}{12}N^3 + \frac{1}{8}N^4 : N \ge 1 \right\}$$

References I

[1] Péricles Alves, Fabian Gruber, Johannes Doerfert, Alexandros Lamprineas, Tobias Grosser, Fabrice Rastello, and Fernando Magno Quintão Pereira. "Runtime Pointer Disambiguation". In: *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. OOPSLA 2015. Pittsburgh, PA, USA: ACM, 2015, pp. 589–606. DOI: 10.1145/2814270.2814285.

[2] Riyadh Baghdadi, Albert Cohen, S. V., and Konrad Trifunovic. "Improved loop tiling based on the removal of spurious false dependences". In: *TACO* 9.4 (2013), p. 52. DOI: 10.1145/2400682.2400711.

References II

[3] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. “The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems”. In: *Science of Computer Programming* 72.1–2 (2008), pp. 3–21. DOI: 10.1016/j.scico.2007.08.001.

[4] Denis Barthou, Albert Cohen, and Jean-François Collard. “Maximal static expansion”. In: *POPL ’98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. San Diego, California, United States: ACM, 1998, pp. 98–106. DOI: 10.1145/268946.268955.

[5] Denis Barthou, Jean-François Collard, and Paul Feautrier. “Fuzzy Array Dataflow Analysis”. In: *J. Parallel Distrib. Comput.* 40.2 (1997), pp. 210–226. DOI: 10.1006/jpdc.1996.1261.

[6] Cédric Bastoul. *Generating loops for scanning polyhedra*. Tech. rep. 2002/23. Versailles University, 2002.

References III

[7] Cédric Bastoul. *Extracting polyhedral representation from high level languages*. Tech. rep. LRI, Paris-Sud University, May 2008.

[8] Marouane Belaoucha, Denis Barthou, Adrien Eliche, and Sid-Ahmed-Ali Touati. “FADAlib: an open source C++ library for fuzzy array dataflow analysis”. In: *Intl. Workshop on Practical Aspects of High-Level Parallel Programming*. Vol. 1. 1. Amsterdam, The Netherlands, May 2010, pp. 2075–2084. DOI: DOI:10.1016/j.procs.2010.04.232.

[9] Wlodzimierz Bielecki, Marek Palkowski, and Piotr Skotnicki. “Generation of parallel synchronization-free tiled code”. In: *Computing* (Oct. 2017). DOI: 10.1007/s00607-017-0576-3.

References IV

[10] Uday Bondhugula, Muthu Baskaran, Sriram Krishnamoorthy, J. Ramanujam, Atanas Rountev, and P. Sadayappan. “Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model”. In: *International Conference on Compiler Construction (ETAPS CC)*. Apr. 2008. DOI: 10.1007/978-3-540-78791-4_9.

[11] *Candl*. http://icps.u-strasbg.fr/~bastoul/development/candl/.

[12] Alain Darte, Robert Schreiber, and Gilles Villard. “Lattice-Based Memory Allocation”. In: *IEEE Trans. Comput.* 54.10 (2005), pp. 1242–1257. DOI: 10.1109/TC.2005.167.

[13] Paul Feautrier. “Array expansion”. In: *ICS ’88: Proceedings of the 2nd international conference on Supercomputing*. St. Malo, France: ACM Press, 1988, pp. 429–441. DOI: 10.1145/55364.55406.

References V

[14] Paul Feautrier. “Dataflow analysis of array and scalar references”. In: *International Journal of Parallel Programming* 20.1 (1991), pp. 23–53. DOI: 10.1007/BF01407931.

[15] Paul Feautrier. “Some Efficient Solutions to the Affine Scheduling Problem. Part II. Multidimensional Time”. In: *International Journal of Parallel Programming* 21.6 (Dec. 1992), pp. 389–420. DOI: 10.1007/BF01379404.

[16] Tobias Grosser, Armin Größlinger, and Christian Lengauer. “Polly - Performing polyhedral optimizations on a low-level intermediate representation”. In: *Parallel Processing Letters* 22.04 (2012). DOI: 10.1142/S0129626412500107.

[17] Bastian Hagedorn, Larisa Stoltzfus, Michel Steuwer, Sergei Gorlatch, and Christophe Dubach. “High performance stencil code generation with Lift”. In: *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. CGO 2018. Vienna, Austria: ACM, 2018, pp. 100–112. DOI: 10.1145/3168824.

References VI

[18] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. *New user interface for Petit and other interfaces: user guide*. Tech. rep. Available as `petit/doc/petit.ps` in the `Omega` distribution. University of Maryland, Dec. 1996.

[19] Wayne Kelly, Vadim Maslov, William Pugh, Evan Rosser, Tatiana Shpeisman, and David Wonnacott. *The Omega Library*. Tech. rep. University of Maryland, Nov. 1996.

[20] *The Polyhedral Compiler Collection*. `http://www.cse.ohio-state.edu/~pouchet/software/pocc/`. 2012.

[21] Louis-Noël Pouchet, Cédric Bastoul, and Albert Cohen. *LetSee: the LEgal Transformation SpacE Explorator*. Third International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES'07), L'Aquila, Italia. Extended abstract, pp 247–251. July 2007.

References VII

[22] Louis-Noël Pouchet, Uday Bondhugula, Cédric Bastoul, Albert Cohen, J. Ramanujam, P. Sadayappan, and Nicolas Vasilache. "Loop Transformations: Convexity, Pruning and Optimization". In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '11. Austin, TX, Jan. 2011, pp. 549–562. DOI: `10.1145/1926385.1926449`.

[23] Konrad Trifunovic, Albert Cohen, David Edelsohn, Feng Li, Tobias Grosser, Harsha Jagasia, Razya Ladelsky, Sebastian Pop, Jan Sjödin, and Ramakrishna Upadrasta. "GRAPHITE two years after: First lessons learned from real-world polyhedral compilation". In: *GCC Research Opportunities Workshop (GROW'10)*. 2010.

References VIII

[24] S. V. "isl: An Integer Set Library for the Polyhedral Model". In: *Mathematical Software - ICMS 2010*. Ed. by Komei Fukuda, Joris Hoeven, Michael Joswig, and Nobuki Takayama. Vol. 6327. Lecture Notes in Computer Science. Springer, 2010, pp. 299–302. DOI: `10.1007/978-3-642-15582-6_49`.

[25] S. V. "Polyhedral process networks". In: *Handbook of Signal Processing Systems*. Ed. by Shuvra Bhattacharyya, Ed Deprettere, Rainer Leupers, and Jarmo Takala. Springer, 2010, pp. 931–965. DOI: `10.1007/978-1-4419-6345-1_33`.

[26] S. V. "Counting Affine Calculator and Applications". In: *First International Workshop on Polyhedral Compilation Techniques (IMPACT'11)*. Chamonix, France, Apr. 2011. DOI: `10.13140/RG.2.1.2959.5601`.

References IX

[27] S. V. *PENCIL support in pet and PPCG*. Tech. rep. RT-457, version 2. INRIA Paris-Rocquencourt, May 2015. DOI: `10.13140/RG.2.1.4063.7926`.

[28] S. V. *Presburger Formulas and Polyhedral Compilation*. 2016. DOI: `10.13140/RG.2.1.1174.6323`.

[29] S. V. and Albert Cohen. "Live-Range Reordering". In: *Proceedings of the sixth International Workshop on Polyhedral Compilation Techniques*. Prague, Czech Republic, Jan. 2016. DOI: `10.13140/RG.2.1.3272.9680`.

[30] S. V. and Tobias Grosser. "Polyhedral Extraction Tool". In: *Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12)*. Paris, France, Jan. 2012. DOI: `10.13140/RG.2.1.4213.4562`.

References X

[31] S. V., Serge Guelton, Tobias Grosser, and Albert Cohen. “Schedule Trees”. In: *Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques*. Vienna, Austria, Jan. 2014. DOI: 10.13140/RG.2.1.4475.6001.

[32] S. V., Juan Carlos Juega, Albert Cohen, José Ignacio Gómez, Christian Tenllado, and Francky Catthoor. “Polyhedral parallel code generation for CUDA”. In: *ACM Trans. Archit. Code Optim.* 9.4 (2013), p. 54. DOI: 10.1145/2400682.2400713.

[33] S. V., Hristo Nikolov, and Todor Stefanov. “On Demand Parametric Array Dataflow Analysis”. In: *Third International Workshop on Polyhedral Compilation Techniques (IMPACT’13)*. Berlin, Germany, Jan. 2013. DOI: 10.13140/RG.2.1.4737.7441.

[34] Nicolas Vasilache. “Scalable Program Optimization Techniques in the Polyhedral Model”. PhD thesis. Université Paris Sud XI, Orsay, Sept. 2007.

References XI

[35] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S. Moses, S. V., Andrew Adams, and Albert Cohen. “Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions”. In: *ArXiv e-prints* (Feb. 2018). arXiv: 1802.04730 [cs.PL].

[36] Doran K. Wilde. *A Library for doing polyhedral operations*. Tech. rep. 785. IRISA, Rennes, France, 1993, 45 p.